Optimizing Protocol Rewrite Rules of CIL Specifications*

G. Denker†, J. Millen†, A. Grau‡, J. Küster Filipe‡
† Computer Science Laboratory, SRI International, Menlo Park, CA 94025, USA
{denker,millen}@csl.sri.com
‡ Institut für Software, Technische Universität Braunschweig,
Postfach 3329, 38023 Braunschweig, Germany
{antonio.grau,j.kuester-filipe}@tu-bs.de

Abstract

For purposes of security analysis, cryptographic protocols can be translated from a high-level message-list language like CAPSL into a multiset rewriting (MSR) rule language like CIL. The natural translation creates two rules per message or computational action. We show how to optimize the natural rule set by about 50% into a form similar to the result of hand encoding, and prove that the transformation is sound because it is attack-preserving, and unique because it is terminating and confluent. The optimization has been implemented in Java.

1 Introduction

Cryptographic protocol analysis is often done from a state-transition model of the protocol, whether the approach involves inductive proofs or a state-space search. While the mathematical representations of these models vary considerably across different approaches, from process algebras through trace models to strand spaces, there is a great deal of commonality in the abstract symbolic treatment of protocols and the associated Dolev-Yao-based attacker model. This commonality was recognized and codified in the Cervasato, *et al* meta-notation, in which state transitions are expressed with multiset rewriting (MSR) rules [CDL+99].

While the MSR model is suitable as a basis for certain kinds of analysis and perhaps as a common language that can be translated automatically

*Supported by DARPA through Air Force Research Laboratory under Contract F30602-98-C-0258.

into other representations, it is still far from the kind of high-level specification language that would be seen by protocol designers as a natural form in which to express their algorithms. In journal articles, textbooks, and specification documents, protocol designers normally use an informal notation to summarize the structure and sequence of messages. An example of a message in this style is:

$$A \to B : \{A, N\}_K.$$

This is a message from A to B with one encrypted field. The field is the concatenation of A with another field N and the encryption key is K. The cryptosystem and other details must be determined from accompanying text.

Message-list notations have been used as formal specification languages for certain tools, which translate them into other forms suitable for analysis. This was done by Carlsen, who generated per-process CKT5 specifications from "Standard Notation" [Car94]; by Brackin, who produced HOL statements from ISL, which includes annotations to support authentication logic [Bra97]; and by Lowe, whose Casper language is translated into a dialect of CSP [Low98]. Our own effort centers on the language CAPSL (Common Authentication Protocol Specification Language) [DM99, DM00], which, in an attempt to be truly universal, is translated into a dialect of MSR, called CIL (CAPSL Intermediate Language). The idea of having a single common protocol specification language that could be used as the input format for any formal analysis technique was first presented at the 1996 Isaac Newton Institute Programme on Computer Security, Cryptology, and Coding Theory at Cambridge University, where an early version of CAPSL was presented. The current CAPSL project includes an effort to produce translators called connectors from CIL to existing and new analysis tools.

The basic, natural translation from CAPSL to CIL, as described in [DM99], generates two rewrite rules per message, one for the message sender and one for the message receiver. Often, however, the transition that receives a message and the one from the same agent that sends a reply can be collapsed into a single transition that does both, and MSR protocol encodings produced by hand usually have this characteristic. Successive computations by the same agent to update or enlarge its state memory can also be combined.

The optimization algorithm described in this paper automatically implements the kind of rule combinations that would typically be done by hand. Relative to the simple message-by-message translation, this reduces the number of rules, as well as the number of states per role, by about

50%. We show that this reduction is sound in the sense that it is attack-preserving, by essentially the same definition used in [SS98]. The optimizing transformation has been implemented as a post-processing step in the CAPSL translator.

The number of rules has direct impact on the performance of state evaluation tools such as model checkers. In the model-checking approach, a finite instantiation of the protocol is tested for security breaches. For this purpose, an exhaustive search strategy enumerates all reachable states for a given initial state and tests whether they invalidate a given security property. Even for small protocols and very restricted numbers of sessions the number of states explodes. This is due partly to the fact that the intruder behavior is highly non-deterministic, and partly due to the fact that new sessions involving legitimate principals may be created and execute asynchronously. Thus, a linear reduction in the number of rules can reduce the number of states to be explored by an exponential factor.

Because optimizations are performed as a series of successive rule-combination steps, there is a question as to whether the order of combination steps affects the size of the final set of rules. We show that the optimization, considered as a reduction system, is terminating and confluent, and hence canonical, so that the final set of rules is unique.

In Section 2, we review the MSR representation of cryptographic protocols, and the form of it used in CIL. Section 3 gives some examples of optimization steps and then states it formally. The soundness of the optimization is shown in Section 4, and uniqueness is proved in Section 5. Finally, we discuss the implementation in Section 6.

2 CIL Representation of Messages

We begin by describing the MSR model and then show how it has been instantiated as CIL.

2.1 The MSR Operational Model

As defined in [CDL+99], MSR rules are of the form

$$F_1 \dots F_k \rightarrow (\exists x_1, ..., x_m)\ G_1 \dots G_n,$$

where each F_i and G_j is a "fact." Facts are atomic formulas of the form $P(t_1, ..., t_r)$ where P is a predicate symbol and the arguments t_i are terms. A term is constructed from constants, variables, and function symbols. Free variables are implicitly universally quantified. Existentially quantified variables are treated specially in this context: they are instantiated with fresh (unused) constants, and hence are used to model nonce generation.

The global state of a system is represented as a multiset of facts. A multiset is a set in which there may be more than one copy of each element. A particular system state would have only ground facts (with no variables).

Rewrite rules specify state transitions. A rule is eligible to fire when the facts on the left side of the rule can be matched with facts in the multiset. When a rule fires, the matching facts in the multiset are removed from it and replaced by the facts on the right side of the rule, instantiated according to the substitution required by the pattern match. Removing a fact from the multiset reduces its multiplicity by one, if it was more than one. Facts in the multiset are typically ground terms (no variables) when finite-state search tools are used.

A rule may be expressed in an abbreviated form as:

$$\mathbf{F} \to (\exists \mathbf{V})\ \mathbf{G}$$

where $\mathbf{F}$, $\mathbf{G}$ are (possibly empty) multisets of facts, and $\mathbf{V}$ is a set of variables. An individual fact may also be treated as a singleton multiset with one occurrence of that fact. We use juxtaposition to combine multisets, so that $\mathbf{FG}$ is the multiset with the elements of $\mathbf{F}$ and $\mathbf{G}$, with the sum of their multiplicities in $\mathbf{F}$ and $\mathbf{G}$. The quantification $(\exists \mathbf{V})$ is omitted if $\mathbf{V}$ is empty.

2.2 Protocol Rules

When the MSR notation is applied to specify protocols, a few special kinds of facts are used, to express the local states of protocol processes (which we call *agents*), messages, and intruder knowledge. CIL also has additional kinds of facts to help express security goals. The MSR version of the global state facts are:

(local) state $Role_i(t_1, ..., t_n)$

message $M(a, b, t)$

intruder memory $N(t)$

where t is a term, a and b are principals, i is a state label (usually an integer), and *Role* is any of the principal names occurring in the protocol specification, representing roles. For example, $A_0(\texttt{Alice}, \texttt{Bob})$ is the initial state of an initiator Alice who intends to talk to Bob. Our usage of M and N differs from that in [CDL+99].

In CIL rules, symbols are typed, and predicate symbols have signatures. Thus, for example, a in $M(a,b,t)$ is always of type Principal. The signature of a local state predicate depends on the state.

Rules with an empty left side are interpreted as initialization or fact-generating rules. For each role in the protocol, an initial state fact is generated with initially held variables. The rule

$$\rightarrow A_0(A,B)\ B_0(B)$$

creates two facts representing the initial states of agents playing the two roles in a protocol, one in role "A" and the other in role "B". Since A and B are variables of type Principal, this rule can initiate sessions between any pair of principals. In particular, the rule can initiate several sessions involving the same principals; this is one reason why the global state is a multiset rather than a set.

CIL also has its own syntax, consisting of a uniform abstract syntax with a standard functional representation. For example, a state fact $A_0(A,B)$ would appear in CIL as `state(roleA,0,terms(A,B))`, and a rule has the form `rule(facts(`*left-facts*`),id(`*vars*`), facts(`*right-facts*`))`. However, the details of the CIL representation are not pertinent here.

A message such as this one, $A \rightarrow B : \{A,N\}_K$, is naturally translated into two transitions, one for the sender A and one for the receiver B. Assuming that A is in its initial state 0 before sending this message, and N is a nonce which is generated in this message, the A transition would be:

$$A_0(A,B) \rightarrow (\exists N)\ A_1(A,B,N)\ M(A,B,\{A,N\}_K).$$

(The encrypted term would normally be represented functionally, perhaps as $se(K, cat(A,N))$, but the conventional notation is retained here for easy readability. CIL uses the functional notation.) Since each message results in two CIL rules, the CIL specification typically has twice as many rules as messages, plus a few other special rules such as for initialization.

In CAPSL specifications, equational actions for tests and computations may be interleaved with messages. Actions also result in transition rules. If an equation has an uninitialized variable on the left, it is an assignment statement and it causes the local state of some agent to be expanded. For example, the action $K_1 = \{A,N\}_K$ in a phrase executed by A in state 1 would be translated into the rule:

$$A_1(A,B,N) \rightarrow A_2(A,B,\{A,N\}_K).$$

However, if A had already learned or computed a value for K_1, the same equation would be treated as a test, and it would generate the two rules:

$$A_1(A, B, K_1) \rightarrow A_2(A, B, K_1, K_1 = \{A, N\}_K)$$
$$A_2(A, B, K_1, \mathit{true}) \rightarrow A_3(A, B, K_1).$$

Besides the honest agents, the intruder or attacker also has to be formally described. The Dolev-Yao model of the attacker, commonly used in protocol analysis, can be expressed as a set of rules that allow the attacker to eavesdrop on transmitted messages and to forge new messages from available message components. The attacker can also decompose and construct terms to expand its memory.

Eavesdropping is implemented by the rule:

$$M(A, B, T) \rightarrow N(T)$$

Message introduction is implemented by the rule:

$$N(T) \rightarrow M(\mathit{Spy}, A, T)$$

where *Spy* is the attacker's identity as a principal, and computations are expressed with rules of the form:

$$\mathbf{N} \rightarrow N(T)$$

where $\mathbf{N}$ is a multiset of N facts. The new term T is computed from the N facts using public functions and inversions to compose, decompose, encrypt, decrypt, etc. An example of a rule of this kind is:

$$N(K)\ N(T) \rightarrow N(\{T\}_K).$$

This set of attacker rules is intended more to define the attacker rather than as a concrete recommendation for model-checking. For model-checking there is an advantage to separating composition and decomposition activities, using distinct memory predicates [CDL+99, CJM98]. The current CAPSL translator does not generate attacker rules, but we need to make assumptions about them to prove that our optimizations are attack-preserving.

3 Optimization Examples and Steps

In order to motivate and explain optimization, we use an example to show the natural correspondence between protocol messages and CIL rules.

3.1 CAPSL Example

Here is the CAPSL specification of the Needham-Schroeder Public-Key protocol, in CAPSL notation.

```
PROTOCOL NSPK;
VARIABLES
    A, B: PKUser;
    Na, Nb: Nonce, CRYPTO;
ASSUMPTIONS
    HOLDS A: B;
MESSAGES
  A -> B: {Na, A}pk(B);
  B -> A: {Na, Nb}pk(A);
  A -> B: {Nb}pk(B);
GOALS
  SECRET Na: A, B;
  SECRET Nb: A, B;
END;
```

This protocol is well enough known to need no explanation, but at least one aspect of the specification deserves comment. The principals A and B are declared to be of type `PKUser` because principals of that subtype have the public-key function `pk` defined for them, as well as the corresponding private-key function `sk`. There is an abstract datatype specification defining this subtype in the CAPSL prelude, so it does not have to appear explicitly.

3.2 Optimization Steps

Given this NSPK specification, the following two CIL rules represent B's receipt of the first message and B's sending of the second message.

$$\begin{aligned}
&rule1: \ B_0(B)\ M(X, B, \{N_a, A\}_{pk(B)}) \rightarrow B_1(B, N_a, A)\\
&rule2: \ B_1(B, N_a, A) \rightarrow (\exists N_b)\ B_2(B, N_a, A, N_b)\ M(B, A, \{N_a, N_b\}_{pk(A)})
\end{aligned}$$

Under the assumption that agents have a deterministic behavior, i.e., at most one rule is applicable in each agent state, we know that after the receiving the message from A, the only thing B can do is to reply with the second message to A. The following optimized rule combines B's behavior into a one-step transition in which B receives A's message and immediately replies to it:

$$\begin{aligned}
rule1,2: \ & B_0(B)\ M(X, B, \{N_a, A\}_{pk(B)}) \rightarrow\\
& (\exists N_b)\ B_2(B, N_a, A, N_b)\ M(B, A, \{N_a, N_b\}_{pk(A)})
\end{aligned}$$

When the two rules are combined, the original pair of rules is deleted. Optimization occurs only when there is no other way to enter state B_1, so in effect state B_1 is also eliminated.

Combining the rules in this example is straightforward since the right-hand side of the first rule and the left-hand side of the second rule are identical. More generally, for two rules R and R' to be optimizable it is necessary (though not sufficient) that the state fact on the right-hand side of R is an instantiation of the state fact on the left-hand side of R'. The next example illustrates this.

For the sake of this example, replace the message `B -> A: {Na,A}pk(B)` in NSPK by a sequence of two actions, an assignment and a message transmission, so that the message list is:

```
MESSAGES
  A -> B: {Na, A}pk(B);
   T = {Na, Nb};
  B -> A: {T%{Na,Nb}}pk(A);
  A -> B: {Nb}pk(B);
```

(The % is Lowe's operator, where $U\%V$ is a field viewed by the sender as U and by the receiver as V.)

This message list yields the following CIL rules for B transitions.

$$
\begin{aligned}
&rule1: \ B_0(B)\ M(X, B, \{N_a, A\}_{pk(B)}) \rightarrow B_1(B, N_a, A)\\
&rule2: \ B_1(B, N_a, A) \rightarrow (\exists N_b)\ B_2(B, N_a, A, N_b, \{N_a, N_b\})\\
&rule3: \ B_2(B, N_a, A, N_b, T) \rightarrow B_3(B, N_a, A, N_b, T)\ M(B, A, \{T\}_{pk(A)})
\end{aligned}
$$

In this case, besides combining $rule1$ and $rule2$, we can also combine $rule2$ and $rule3$, since $B_2(B, N_a, A, N_b, T)$ can be instantiatiated to $B_2(B, N_a, A, N_b, \{N_a, N_b\})$ with the substitution $T \mapsto \{N_a, N_b\}$. Thus, we can optimize these rules to

$$
\begin{aligned}
rule2,3: \ & B_1(B, N_a, A) \rightarrow\\
& \quad (\exists N_b)\ B_3(B, N_a, A, N_b, \{N_a, N_b\})\ M(B, A, \{N_a, N_b\}_{pk(A)}).
\end{aligned}
$$

3.3 Attack Preservation

In order to assure that our optimization technique is attack-preserving, we need to make further restrictions on the form of optimizable rules. For a pair (R, R') of rewrite rules to be combined, we require that the second rule has no messages on the left-hand side. We show with the help of a simplified example that allowing a message on the left-hand side of the second rule is unsafe.

Assume the following two rewrite rules for an agent in role B.

$$rule1: \ B_0(B) \ \rightarrow B_1(B,B)$$
$$rule2: \ B_1(B,B) \ M(A,B,sk(B)) \rightarrow B_2(B,B)$$

Since the state predicate $B_1(B,B)$ occurs in both $rule1$ and $rule2$ one might be tempted to optimize these rules to

$$rule1,2: \ B_0(B) \ M(A,B,sk(B)) \rightarrow B_2(B,B).$$

Assume furthermore that $M(A,B,sk(B))$ is an *impossible* message, because B never transmits $sk(B)$, and that B_1 is a state in which a protocol invariant fails, perhaps because it requires that the first two components of B's state must be different. Thus, the failure state is reachable in the original protocol specification, but since B_1 has been deleted by optimization and B_2 cannot be reached since M is impossible, the attack state is no longer present in the optimized protocol. This is why the left-hand side of the second rule is not allowed to have messages.

3.4 Name clashes

Before we can formally define the optimization of two rewrite rules, we have to deal with variable name clashes. In order to avoid accidentally introducing bindings between variables, we apply renaming functions. The following example illustrates the need for renaming.

Assume the following two rules which accidentally use the same variable X:

$$rule1: \ B_0(B) \ M(X,B,A) \rightarrow B_1(B,A)$$
$$rule2: \ B_1(B,A) \ \rightarrow (\exists X) \ B_2(B,A,X).$$

These rules are optimizable. The variable X is used in both rules, though there is no relation between the variable X of $rule1$ and the variable X of $rule2$. To avoid introducing a binding between these two independent variables, we rename X of $rule2$ to X' in the optimized rule. Thus, the optimized rule for $rule1$ and $rule2$ is

$$rule1,2: \ B_0(B)M(X,B,A) \rightarrow (\exists X') \ B_2(B,A,X').$$

The coincidence of variables B and A in the two rules is not a problem because the need to unify the B_1 facts in the two rules determines the appropriate substitution for them.

3.5 Formal Definitions

As intuitively illustrated in the previous section some restrictions on rules are necessary to guarantee an attack-preserving optimization. In summary, we only consider rules that describe asynchronously communicating, deterministically behaving agents, where each agent state is generated by at most one rule.

For optimizations we deal only with local rules, in which only one state fact appears on the left and one on the right of the rule. The rules that arise from protocol transitions in an asynchronous environment are normally local, since only one agent changes state at a time.

Definition 1 *A rule is* **local** *if it is of the form*

$$R : F \ \mathbf{M} \rightarrow (\exists \mathbf{V}) \ F' \ \mathbf{M}'$$

where F, F' are state facts for the same role, and $\mathbf{M}$ and $\mathbf{M}'$ contain no state facts.

States that are optimized away have to be deterministic in both directions. The first rule of an optimized pair needs a backward deterministic state on the right, while the second rule needs a forward deterministic state (the same state) on the left.

Definition 2 *A state A_i is* **forward deterministic** *in $\mathcal{R}$ if there exists at most one rule in $\mathcal{R}$ with a fact of that state on its left-hand side. A local rule is forward deterministic if its state is forward deterministic.*

A state A_i is **backward deterministic** *in $\mathcal{R}$ if there exists at most one rule in $\mathcal{R}$ with a fact of that state on its right-hand side. A local rule is backward deterministic if its state is backward deterministic.*

A state A_i is **deterministic** *in $\mathcal{R}$ if it is both forward and backward deterministic.*

In the following definition of optimizable pairs of rules, $Vars(G)$ is the set of variables occuring in G.

Definition 3 *Given a pair of local rules (R, R') in $\mathcal{R}$ of the form:*

$$R : F \ \mathbf{M_1} \rightarrow (\exists \mathbf{V_1}) \ G \ \mathbf{M_2}$$
$$R' : \ G' \rightarrow (\exists \mathbf{V_2}) \ H \ \mathbf{M_3}$$

*Then the pair (R, R') is σ-***optimizable** *if*

1. *R and R' are local on the same role*

2. *There are no variable name clashes between R and R'*

3. *There exists a substitution σ on $Vars(G')$ such that $G'/\sigma = G$*

4. *The state of which G is a fact is deterministic.*

As mentioned before, name clashes have to be resolved before our optimization technique is applied. Name clashes can always easily be resolved by renaming variables. In section 5 we describe how variable renaming can be efficiently realized for CIL.

We now can present the definition of the optimized rule for an optimizable pair of rules.

Definition 4 *Given a pair (R, R') of σ-optimizable protocol rewrite rules of the form:*

$$R : F\ \mathbf{M_1} \rightarrow (\exists \mathbf{V_1})\ G\ \mathbf{M_2}$$
$$R' :\ G' \rightarrow (\exists \mathbf{V_2})\ H\ \mathbf{M_3}$$

then an **optimization step** *removes R, R' from $\mathcal{R}$ and replaces them with $R^o = opt(R, R')$, defined as*

$$R^o : F\ \mathbf{M_1} \rightarrow (\exists \mathbf{V_1}\ \mathbf{V_2})\ \mathbf{M_2}\ (H\ \mathbf{M_3})/\sigma.$$

4 Properties of Optimization

We show that optimization is sound in the sense that it is attack-preserving. We also show that, under additional assumptions, it delivers a unique set of optimized rewrite rules regardless of the order in which optimization steps are applied.

Protocol Security Invariants
Before we go into the details of the proof, we make some observations about protocol properties. Like Shmatikov and Stern [SS98], we only deal with protocol security properties that are invariants; that is, they are properties of the global state that are supposed to hold for all reachable states.

Furthermore, the invariants depend only on state facts and intruder memory, not on message facts. Secrecy invariants state that the intruder memory does not contain certain terms (which appear in the state memory of some honest principal), and other security properties such as agreements and precedence refer only to state facts. As pointed out in [SS98], if a security invariant is false, it remains false if the intruder's knowledge increases. They called this property *monotonicity of invariance.*

We make use of a more general characterization of security invariants. In protocols that can be expressed in CAPSL and translated to CIL, state memory is monotonic for honest principals as well. Once an honest agent holds a value for a protocol variable (associated with an argument position in its state memory), that value never changes. It follows that invariants that depend only on the global state cannot change their truth value once the relevant variables have become defined for a given agent. In particular, if a state invalidates a protocol invariant, every successor state will violate this invariant as well. We refer to this property as *persistence of violations.*

4.1 Soundness

Our soundness argument reasons about the *state graph* (S, T) of a rule set. The nodes of this state graph are the possible global states (multisets of ground facts) S of the protocol. The graph has directed, labelled transitions (edges) T consisting of pairs of states related by the instantiation of a rule, which labels the transition. A state is *reachable* if there is a sequence of transitions to it from the empty multiset, which is the initial state. We will refer to a state graph simply by its transition set T, since one can find all reachable states in it.

For example, the transition $s \overset{R/\alpha}{\rightarrow} s'$ means that there exists a rule R : $\mathbf{F} \rightarrow (\exists \mathbf{V})\ \mathbf{G}$ and a substitution α such that $\mathbf{F}/\alpha$ is a subset of s and the resulting system state s' is derived from s by replacing the multiset of ground terms $\mathbf{F}/\alpha$ with the multiset of ground terms $\mathbf{G}/\alpha$. (The substitution α assigns unused values to the variables in $\mathbf{V}$.)

Optimization steps eliminate two rules and replace them with a new combined rule. This changes the state graph by eliminating those transitions labelled with instantiations of the eliminated rules, and adding new transitions made possible by the new combined rule. The new state graph has the same set of states, but some of the states have become unreachable because some local states have been optimized away.

Definition 5 *An optimization step taking T to T' is* **attack-preserving** *if, for any security invariant φ, and any state s reachable in T that violates φ, there is a state s' reachable in T' that also violates φ.*

Theorem 1 *Optimization steps are attack-preserving.*

The following lemma is helpful in order to prove Theorem 1. It shows that a transition instantiating the second rule R' of an optimizable pair (R, R') commutes with any other transition in T.

Lemma 1 *Let T be the state graph of $\mathcal{R}$. Let $R' : G' \to (\exists \mathbf{V_2})\ H\ \mathbf{M_3}$ be a local rule such that the state on the left is forward deterministic. Then, an R' transition forward commutes with any other transition in T. That is, if $s_1 \overset{R'/\alpha}{\to} s_2$ and $s_1 \overset{\hat{R}/\hat{\alpha}}{\to} s_3$ then there exists s_4 such that $s_2 \overset{\hat{R}/\hat{\alpha}}{\to} s_4$ and $s_3 \overset{R'/\alpha}{\to} s_4$.*

$$\begin{tikzcd} s_1 \arrow[r, "\hat{R}/\hat{\alpha}"] \arrow[d, "R'/\alpha"'] & s_3 \arrow[d, "R'/\alpha"] \\ s_2 \arrow[r, "\hat{R}/\hat{\alpha}"'] & s_4 \end{tikzcd}$$

Figure 1: Commutativity

Proof:
It suffices to show that the left sides of the two instantiated rules R'/α and $\hat{R}/\hat{\alpha}$ are disjoint, since then they can be applied in either order with the same net result. Since the state of G' is forward deterministic, no rule other than R' can be applied to G'/α. If $\hat{R} = R'$, and the substitutions are different, then the two rules apply to disjoint left sides. If $\hat{R}/\hat{\alpha} = R'/\alpha$ then $s_2 = s_3$ and it is not a different transition. $\square$

Proof of Theorem 1:
The proof is given in the Appendix. The basic idea is to show that if a state violating a security invariant is reachable with a path that includes transitions due to one or both of the rules that have been eliminated by the optimization step, then that state is reachable using an alternate path that uses the new rule resulting from the optimization. Sometimes, one cannot reach the original violation state, but then one can reach another state reachable from the violation state, which must be a violation state by the persistence-of-violations assumption.

4.2 Termination and Uniqueness

The motivation for optimization is to reduce the number of states in order to speed up state evaluation tools such as model checkers. Our proposed optimization technique consists of single optimization steps performed in sequence. For analysis tools it is of importance whether the order in which optimization steps are performed has an impact on the final set of rules or on the final number of rules. We will show that optimization is terminating and delivers a unique result.

Optimization can be understood as a rewrite system in which a set of rules (a term) is rewritten to an optimized set of rules. A well known result in the theory of rewrite systems says that a term has a unique normal form (i.e., it cannot be further rewritten into another term) if the rewrite system is canonical (for details see for instance [Sny91]). For a rewrite system to be canonical it has to be noetherian and confluent. Noetherian systems have no infinite sequences of rewrites. A rewrite system is termed confluent when for any term which can be rewritten into two different terms via several rewrite steps, there exists a common reduction term.

We show that our optimization process, understood as a rewrite system tranforming between sets of rules, is canonical. That means that optimization is a terminating process which delivers a unique set of rules as result. Therefore, speaking in terms of the associated state graphs, the original state graph T and the fully optimized state graph T', we can infer that T' is uniquely determined. For practical purposes that means that applying the optimization steps proposed in this paper in any order always leads to the same optimized state graph which can be used for security analysis.

Theorem 2 *Given a state graph T, then there exists a uniquely determined fully optimized state graph T'.*

The following lemmas are helpful to prove termination and uniqueness of optimization. The first one states that a rule is optimizable with at most one rule to the right (in an optimizable pair) and at most one rule to the left.

Lemma 2 *Consider a rule set $\mathcal{R}$ and a local rule $R \in \mathcal{R}$. There exist at most two rules in $\mathcal{R}$ with which R can form optimizable pair of rules. More specifically,*

(right) *there exists at most one rule $R' \in \mathcal{R}$ such that (R, R') is a pair of optimizable rules, and*

(left) *there exists at most one rule $R' \in \mathcal{R}$ such that (R', R) is a pair of optimizable rules*

Proof:
(right) Assume that for a given R there exist two rules R' and R'' such that (R, R') and (R, R'') are optimizable pairs of rules. If

$$
\begin{aligned}
&R : F\ \mathbf{M_1} \rightarrow (\exists \mathbf{V_1})\ G\ \mathbf{M_2} \\
&R' :\ G' \rightarrow (\exists \mathbf{V_2})\ H\ \mathbf{M_3} \\
&R'' :\ G'' \rightarrow (\exists \mathbf{V_3})\ I\ \mathbf{M_4}
\end{aligned}
$$

then Def. 3 requires G, G', G'' to be state facts of the same role, and the state of which G (and therefore G' and G'' are facts) is forward deterministic. This is obviously not the case.
(right) The proof is analogous using backward determinism. □

Using the previous lemma, we can argue that a given set of rules can be arranged into a totally ordered list of rules such that two rules are optimizable only if they are adjacent (but adjacent rules do not necessarily form an optimizable pair). In the following we refer to such a list as list of optimizable rules. We show that optimization steps are locally confluent. That means one can always reach a common list of rules after two optimization steps (generally, local confluence allows for more than two rewrite steps).

If two optimization steps involve different optimizable pairs of rules, then they are commutative. That is, they might be executed in either order and the order has no effect on the resulting list of rules. If two optimization steps have a common rule, then after one optimization step the other optimization step is no longer applicable since the rule which both steps had in common has been deleted. But the new optimized rule can be taken for another optimization step to yield a common list of rules. Moreover, the optimization relation between rules is preserved. In summary, we show that performing optimization steps on a list of optimizable rules satisfies the following two conditions.

1. Performing an optimization step rewrites a list of optimizable rules into another list of optimizable rules. Assume in the original list the pair (R_1, R_2) has been optimized to R^o, then R^o is optimizable to the left with whatever rule R_1 was optimizable to the left, and R^o is optimizable to the right with whatever rule R_2 was optimizable to the right.

2. Moreover, we show that optimization steps are locally confluent. That is, given a list of optimizable rules that can be rewritten into two different list of optimizable rules, we can always perform one more optimization step in order to reach a common list of optimizable rules.

Lemma 3 *Given a list of rules $\vec{R}$ such that two rules form an optimizable pair (R_1, R_2) only if R_2 is successor of R_1 in the list. Then optimization on $\vec{R}$ is locally confluent. That is, if there exist lists of optimizable rules $\vec{R_1}$ and $\vec{R_2}$ such that $\vec{R_1}$ and $\vec{R_2}$ can be derived from $\vec{R}$ by an optimization step, then there exists a list of optimizable rules $\vec{R^o}$ that can be reached from both lists $\vec{R_1}$ and $\vec{R_2}$ by optimization steps.*

$$\begin{tikzcd}
 & \vec{R} \arrow[dl] \arrow[dr] & \\
\vec{R_1} \arrow[dr, dashed] & & \vec{R_2} \arrow[dl, dashed] \\
 & \vec{R^o} &
\end{tikzcd}$$

Figure 2: Local confluence of optimization steps

Proof:
In order to shorten the proof we assume that there are no name clashes between any of the rules in $\vec{R}$. The following proof can be adapted to the case where name clashes are resolved by renaming.

Let $P_1 = (R, R')$ and $P_2 = (R'', R''')$ be two optimizable pairs of rules in $\vec{R}$. We distinguish two cases: (a) the two pairs have no rule in common and (b) the two pairs have one rule in common.

Disjoint pairs: After optimizing one pair of rules, the other pair of rules is still an optimizable pair in the resulting list of rules. Thus, the two optimization steps commute and the resulting list of rules is the same.

Assume (R, R') and (R'', R''') are the two optimizable pairs of rules. One can also easily check that if there was another rule R^a with which one of the eliminated rules formed an optimizable pair, this rule R^a still exists and forms an optimizable pair with the new optimized rule.

Common rule: Now assume $P_1 = (R, R')$ and $P_2 = (R', R'')$. Then

$$\begin{aligned}
&R : F\ \mathbf{M_1} \rightarrow (\exists \mathbf{V_1})\ G\ \mathbf{M_2} \\
&R' :\ G_1 \rightarrow (\exists \mathbf{V_2})\ H\ \mathbf{M_3} \\
&R'' :\ H_1 \rightarrow (\exists \mathbf{V_3})\ I\ \mathbf{M_4}
\end{aligned}$$

and there exist substitutions σ and τ such that $G_1/\sigma = G$ and $H_1/\tau = H$. There are no name clashes between R, R' and R', R''. As mentioned before, to shorten the proof we assume that there are also no name clashes between R, R''. This is not a real restriction since the variables in rules are universally quantified dummy variables that can be renamed.

Optimizing P_1 first yields:

$$R_1^o : F\ \mathbf{M_1} \rightarrow (\exists \mathbf{V_1}\ \mathbf{V_2})\ \mathbf{M_2}\ H/\sigma\ \mathbf{M_3}/\sigma$$
$$R'' :\ H_1 \rightarrow (\exists \mathbf{V_3})\ I\ \mathbf{M_4}.$$

Since $H_1/\tau = H$ it follows that $H_1/(\tau\sigma) = H/\sigma$ where $\tau\sigma$ is the concatenation of substitutions beginning with τ. Also all other requirements of Def. 3 are fullfilled and thus (R_1^o, R'') is a $(\tau\sigma)$-optimizable pair with the resolvent

$$R^o : F\ \mathbf{M_1} \rightarrow (\exists \mathbf{V_1}\ \mathbf{V_2}\ \mathbf{V_3})\ \mathbf{M_2}\ \mathbf{M_3}/\sigma\ I/(\tau\sigma)\ \mathbf{M_4}/(\tau\sigma). \quad (1)$$

Optimizing P_2 first yields:

$$R : F\ \mathbf{M_1} \rightarrow G\ \mathbf{M_2}$$
$$R_2^o : G_1 \rightarrow (\exists \mathbf{V_2}\ \mathbf{V_3})\ \mathbf{M_3}\ I/\tau\ \mathbf{M_4}/\tau.$$

Since $G_1/\sigma = G$ and all other requirements of Def. 3 are fullfilled we conclude that (R, R_2^o) is a σ-optimizable pair with the resolvent R^o as given in (1).

Thus, performing any of the optimization steps first results in a list of rules which can be further optimized to a common list of optimizable rules. Moreover, any previously existing optimization relations between pairs of rules remains, possibly with the new optimized rule instead of the original ones. For instance, if there exists a rule R_1 in $\vec{R}$ such that (R_1, R) is an optimizable pair of rules, than (R_1, R^o) is an optimizable pair of rules in $\vec{R^o}$. This can be checked by applying Def. 3.

□

Proof (Theorem 2):

Since the set of rules is finite and each optimization step reduces the number of rules by one, the optimization process is terminating. Therefore, the optimization process describes a rewrite system that is noetherian. A well known result in the theory of term rewrite system says that a system is canonical if it is noetherian and confluent. A noetherian system is confluent if and only if it is locally confluent. Thus, using Lemma 3 concludes the proof of Theorem 2. □

5 Implementation

A CIL rewrite-rule optimizer has been implemented in Java and is applied as a post-processing step in the CAPSL-CIL translator. It is publicly available together with the CAPSL parser, type-checker and CAPSL-CIL translator at the CAPSL web site [Mil99].

The optimizer starts reading a CIL specification and checks pairs of rewrite rules for optimizability. In order to decide whether two rules are optimizable, the optimizer needs to access information from the CIL specification. In particular, in order to decide whether two state facts are optimization compatible, the types of symbols is checked. This way we can guarantee that a proper substitution mapping between state predicates exists. Moreover, the optimizer needs to access assumptions and goals in order to check that the states to be eliminated are not named in goals and assumptions. As long as two optimizable rules are found, the optimizer computes the optimized rule, deletes the original rules and adds the new optimized one to the rule set.

5.1 Variable Renaming

In previous sections we mentioned the problem of name clashes. The simple-minded solution is to rename all variables in one of the rules in an optimizable pair to new variables. For instance, given the optimizable rules

$$\begin{aligned}&rule1: \; B_0(B,A)\; M(X,B,A) \rightarrow (\exists X) B_1(B,A)\\&rule2: \; B_1(B,A) \rightarrow (\exists X)\; B_2(B,A,X)\end{aligned}$$

we could rename the variables B, A, and X of $rule2$ using the renaming map $B \mapsto B', A \mapsto A', X \mapsto X'$:

$$rule2: \; B_1(B',A') \rightarrow (\exists X')\; B_2(B',A',X')$$

Now, the rules do not have any variable names in common and we may optimize them obtaining

$$rule1,2: \; B_0(B) \; M(X,B,A) \rightarrow (\exists X') \; B_2(B,A,X').$$

As one can observe, some of the renamed variables are mapped back to their original name due to the given substitution map σ. For instance, in the example above B' has been mapped back to B using σ. Thus, a more efficient solution for eliminating name clashes is to only rename those variables which are not mapped by the substitution mapping σ.

Let (R, R') be an optimizable pair of protocol rewrite rules

$$R : F \; \mathbf{M_1} \rightarrow (\exists \mathbf{V_1}) \; P_n(\bar{x}) \; \mathbf{M_2}$$

and

$$R' : P_n(\bar{y}) \rightarrow (\exists \mathbf{V_2}) \; G \; \mathbf{M_3}$$

where $F, G, P_n(\bar{x}), P_n(\bar{y})$ are state facts, $\bar{x} = x_1 \ldots x_r$, $\bar{y} = y_1 \ldots y_r$, and $\mathbf{M_1}, \mathbf{M_2}, \mathbf{M_3}$ are multisets of message facts. Let $\mathbf{W}$ and $\mathbf{W'}$ be the set of variables occurring in R and R' respectively.
The optimum of R and R', R^o, is computed by the following algorithm:

1. $\mathbf{E} = \{x_i \mid x_i = y_i \wedge i = 1, \ldots, r\}$
2. $\mathbf{C} = \{u \mid u \in \mathbf{W} \wedge u \in \mathbf{W'} \wedge u \notin \mathbf{E}\}$
3. $M_{ren} = \{u \mapsto u' \mid u \in \mathbf{C} \wedge u' \notin (\mathbf{W} \cup \mathbf{W'})\}$
4. $R_{ren} = R'/M_{ren}$

   Let $R_{ren} = P_n(\bar{y'}) \; \rightarrow (\exists \mathbf{V'_2}) \; G' \; \mathbf{M'_3}$
5. $M_{subst} = \{y'_i \mapsto x_i \mid y'_i \neq x_i \wedge i = 1, \ldots, r\}$
6. $R^o : F \; \mathbf{M_1} \; \rightarrow (\exists \mathbf{V_1} \; \mathbf{V'_2}) \; \mathbf{M_2} \; (G' \; \mathbf{M'_3})/M_{subst})$
7. The symbol table of the CIL specification is updated with all newly introduced variables.

$\mathbf{C}$ represents the set of variables which may cause a clash. The variables in $\mathbf{C}$ are renamed in R' with new variables using the renaming map M_{ren}. The optimum is now computed using the new rule R_{ren}. At last, new variables are introduced in the symbol table.

Protocol	*# Input Rules*	*# Output Rules*	*Reduction Ratio*
NSPK	7	5	28.57%
EKE	14	6	57.14%
Otway	9	6	33.33%
WMF	5	4	20%
SRP	19	8	57.89%
SSL	29	11	62.07%
Voucher	10	5	50%

Table 1: Reduction ratio of CIL rules

We have applied the optimizer to several protocol specifications. The CAPSL specifications of the protocols may be found at our web site [Mil99]. The CIL specifications were generated using the publicly available CAPSL-CIL translator. Table 1 shows the results. The reduction ratios clearly show that the optimizer may reduce the number of rules significantly. This way, the performance of verification tools, such as finite-state exploration tools, can be drastically increased.

References

[Bra97] S. Brackin. An interface specification language for automatically analyzing cryptographic protocols. In *Symposium on Network and Distributed System Security*. Internet Society, February 1997.

[Car94] U. Carlsen. Generating formal cryptographic protocol specifications. In *IEEE Symposium on Research in Security and Privacy*, pages 137–146. IEEE Computer Society, 1994.

[CDL+99] I. Cervesato, N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. In *12th IEEE Computer Security Foundations Workshop*, pages 55–69. IEEE Computer Society, 1999.

[CJM98] E. Clarke, S. Jha, and W. Marrero. Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In *Proc. IFIP Working Conference on Programming Concepts and Methods (PROCOMET)*, 1998.

[DM99] G. Denker and J. Millen. CAPSL and CIL Language Design: A Common Authentication Protocol Specification Language and Its Intermediate Language. CSL Report SRI-CSL-99-02, Computer Science Laboratory, SRI International, Menlo Park, CA 94025, 1999. `http://www.csl.sri.com/~denker/pub_99.html`.

[DM00] G. Denker and J. Millen. CAPSL Integrated Protocol Environment. In D. Maughan, G. Koob, and S. Saydjari, editors, *Proc. DARPA Information Survivability Conference and Exposition, DISCEX2000, January 25-27, Hilton Head Island, SC, USA*, 2000. `http://schafercorp-ballston.com/discex/`.

[Low98] G. Lowe. Casper: a compiler for the analysis of security protocols. *Journal of Computer Security*, 6(1):53–84, 1998.

[Mil99] J. Millen. CAPSL Web Site. `http://www.csl.sri.com/~millen/capsl`, 1999.

[Sny91] W. Snyder. *A Proof Theory for General Unification.* Birkhäuser, 1991.

[SS98] V. Shmatikov and U. Stern. Efficient Finite State Analysis for Large Security Protocols. In *11th IEEE Computer Security Foundations Workshop, Rockport, Massachusetts, June 1998*, pages 106–115. IEEE Computer Society, 1998.

A Appendix: Proof of Theorem 1

In order to prove that each optimization step is attack preserving, we split an optimization step into substeps:

1. **Adding all new transitions:** An optimization step for the pair (R, R') of optimizable rules introduces several transitions in the state graph T. In particular, if in T a sequence of transitions $s_i \stackrel{R/\alpha}{\rightarrow} s_{i+1} \stackrel{R'/\alpha}{\rightarrow} s_{i+2}$ occurs, then, given the definition of the optimized rule R^o in Def. 3, we can add a transition $s_i \stackrel{R^o/\alpha}{\rightarrow} s_{i+2}$. Adding such transitions does not change the set of states in the graph nor the reachability of states. We refer to the state graph in which all transitions which are instantiations of the optimized rule have been added by T_{+R^o}.

2. **Deleting one transition:** We repeatedly delete one transition labelled R/α or R'/α at a time until no more transitions with those label are present.

We will show that each substep does not interfere with attack preservation.

ad 1. Obviously, the first optimization substep preserves reachability of attack states, since no transitions are deleted in the translation from T to T_{+R^o}.

ad 2. We show inductively that repeatedly deleting a transition $\stackrel{R/\alpha}{\rightarrow}$ or a transition $\stackrel{R'/\alpha}{\rightarrow}$ at a time does not interfere with attack-preservation. The induction invariant is that any given intermediate state graph $T_{+R^o}^{-\mathcal{D}}$ is attack-preserving, where $\mathcal{D}$ is a set of already deleted transitions, i.e., $\mathcal{D} = \{\stackrel{l}{\rightarrow} \mid l = R/\beta \vee l = R'/\beta\}$.

Induction base:
The induction base is T_{+R^o} (no transitions have been removed yet). Obviously, T_{+R^o} is attack-preserving.

Induction hypothesis:
Let $T_{+R^o}^{-\mathcal{D}}$ be an attack-preserving intermediate state graph in which some transitions have been removed.

Induction step:
Let $\stackrel{R_1/\alpha}{\rightarrow}$ be the next transition to be removed. We show that going from $T_{+R^o}^{-\mathcal{D}}$ to $T_{+R^o}^{-(\mathcal{D} \cup \{\stackrel{R_1/\alpha}{\rightarrow}\})}$ preserves attacks. Let φ be a protocol invariant. Let s_v be a reachable state in $T_{+R^o}^{-\mathcal{D}}$ which violates φ.

Case 1: Let $\stackrel{R_1/\alpha}{\rightarrow} = \stackrel{R/\alpha}{\rightarrow}$.

Case 1a: The transition $\stackrel{R/\alpha}{\rightarrow}$ which is to be deleted is not on the path to s_v.
Obviously, deleting this transition does not conflict with reachability of s_v.

Case 1b: The transition $\stackrel{R/\alpha}{\rightarrow}$ which is to be deleted is on the path to s_v, but there is no transition $\stackrel{R'/\alpha}{\rightarrow}$ on the path to s_v.

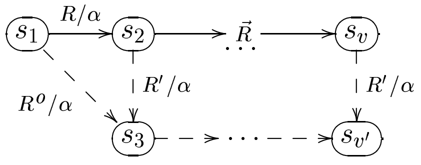

Figure 3: Case 1b

Fig. 3 depicts the situation. Since R and R' are local on the same role, we conclude that in s_2 the transition $\stackrel{R'/\alpha}{\rightarrow}$ is enabled. Thus, there exists a state s_3 such that $s_2 \stackrel{R'/\alpha}{\rightarrow} s_3$. Moreover, given the definition of the optimized rule in Def. 4 we can infer that $s_1 \stackrel{R^o/\alpha}{\rightarrow} s_3$ and this transition is in the state graph $T^{-\mathcal{D}}_{+R^o}$. Recursively applying Lemma 1 (forward commutativity) for the transition $s_2 \stackrel{R'/\alpha}{\rightarrow} s_3$ and all transitions in $\vec{R}$ yields that that there exists a state $s_{v'}$, such that $s_v \stackrel{R'/\alpha}{\rightarrow} s_{v'}$ and also $s_3 \stackrel{\vec{R}}{\rightarrow} s_{v'}$. Because of preservation of violations, $s_{v'}$ also violates φ. Obviously, $s_{v'}$ is still reachable after deleting $\stackrel{R/\alpha}{\rightarrow}$.

Case 1c: Both transitions, $\stackrel{R/\alpha}{\rightarrow}$ and $\stackrel{R'/\alpha}{\rightarrow}$ are on the paths to s_v.

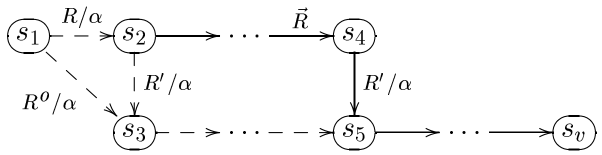

Figure 4: Case 1c

Fig. 4 illustrates this situation. In case there are several transitions $\stackrel{R'/\alpha}{\rightarrow}$ on the path to s_v, choose the one which is closest to the transition $\stackrel{R/\alpha}{\rightarrow}$ that is to be deleted, i.e., $\vec{R}$ does not contain R'/α. Analogously to case 1b we can infer that there exists a state s_3 and transitions $s_2 \stackrel{R'/\alpha}{\rightarrow} s_3$ and $s_1 \stackrel{R^o/\alpha}{\rightarrow} s_3$ in $T^{-\mathcal{D}}_{+R^o}$. Again we apply Lemma 1 (forward commutativity) to the transition $s_2 \stackrel{R'/\alpha}{\rightarrow} s_3$ and all iteratively all transitions in $\vec{R}$. Thus, there exists a state s_5 which is also reachable from s_3 via the transitions in $\vec{R}$. Thus, after deleting the transition $s_1 \stackrel{R/\alpha}{\rightarrow} s_2$ state s_v is still reachable.

Case 2: Let $\stackrel{R_1/\alpha}{\rightarrow} = \stackrel{R'/\alpha}{\rightarrow}$.

Case 2a: The transition $\stackrel{R'/\alpha}{\rightarrow}$ which is to be deleted is not on the path to s_v.
Obviously, deleting this transition does not conflict with reachability of s_v.

Case 2b: The transition $\stackrel{R'/\alpha}{\rightarrow}$ which is to be deleted is on the path to s_v.
We first argue that in this case there also must exists a transition labelled R/α previous to the transition $\stackrel{R'/\alpha}{\rightarrow}$.
Since R and R' are optimizable, and since the state of which G is a fact is backward deterministic we can conclude that R/α is the only way to have produced the fact necessary to enable R'/α. Thus, there is a transition $\stackrel{R/\alpha}{\rightarrow}$ on the path to s_v prior to the one that is to be deleted. In case there are several transitions R/α we choose the one that is closest to R'/α. Thus, we are in a similar situation as depicted in Fig. 4, but now we know that $\vec{R}$ does not contain R/α. Let $s_1 \stackrel{R/\alpha}{\rightarrow} s_2$ be the closest transition with label R/α. Then, because of the definition of R, R', and the optimum R^o, we know that a state s_2 and transitions $s_2 \stackrel{R'/\alpha}{\rightarrow} s_3$ and $s_1 \stackrel{R^o/\alpha}{\rightarrow} s_3$ exists in the state graph. Now we can apply Lemma 1 to transition $s_2 \stackrel{R'/\alpha}{\rightarrow} s_3$ and iteratively all transitions in $\vec{R}$ and get that s_5 is also reachable from s_3 via the transitions in $\vec{R}$. Therefore, deleting the transition $s_4 \stackrel{R/\alpha}{\rightarrow} s_5$ does preserves reachability of the attack state s_v.

$\square$